Calling Watson from RPG

Presented by

Scott Klement

http://www.profoundlogic.com

"Artificial intelligence is about replacing human decision making with more sophisticated techniques"

The Agenda

Agenda for this session:

1. What Is Watson?
2. How Do I Use It?
   - The gist of using it
   - Where to find info
   - Scott's examples
3. Watson Examples from RPG
   - Translation
   - Natural Language Understanding
   - Image Recognition

What is Watson?

- IBM's "cognitive" platform
- Aritificial Intelligence (AI)
- Understand things normally understood only by humans
- Originally played "Jeopardy!"
- Set up and customize for your needs.

Available Two Ways

Watson runs on POWER hardware.

- but, software requires IBM's own Watson (linux-based) platform
- Set up by IBM Watson experts

1. Some big companies have set up Watson on-site
   - (working with IBM experts)
2. Anyone can utilitize it via IBM Cloud (formerly: Bluemix)
   - this is the only way I've had the opportunity to try it.

IBM Cloud is a platform as a service (PaaS) that:

- Provides a Web Service API (often just called "API")
- You can run it from any application that can do web service calls
- Available for free or very low cost.
- Pay more if your usage is higher.

Taking a Tour of the Web Site

Home Page for Watson:
https://www.ibm.com/watson/

The Products/Services Available (i.e. interesting part)
https://www.ibm.com/watson/products-services/
https://www.ibm.com/cloud/ai

Signing Up for IBM Cloud (free)

- The above links have "sign up" options at the top

Inside Your IBM Cloud Account

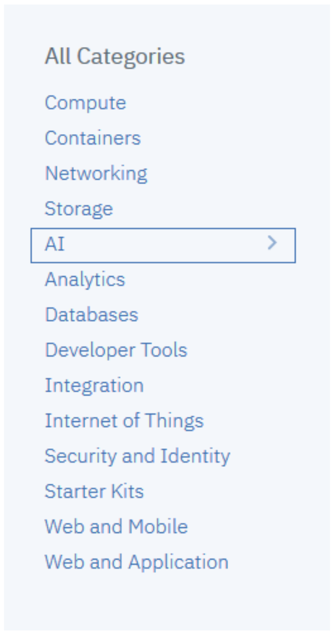
After logging in there is more information

- "Catalog" shows the APIs available
  - Watson APIs are under "AI"
  - IBM Cloud has other (non-Watson) APIs as well

- You can set up a given service to run in your account
- You can read about the APIs, see docs and demos as well

- This is what your RPG programs will actually connect to
- This is what you can use directly.
- Dashboard/apps shows what you currently have set up

Watson Docs / Examples



To read about the different services:

- Go to the catalog link
- In the categories (on the left) click AI

The next page shows what you can "create"

- "create" because you set up your own copy
- On your own (cloud-based) server

If you click one

- Option to create it
- Option to view docs
- Sometimes there's a demo available
- Pricing plan(s)

Create a Service on your IBM Cloud

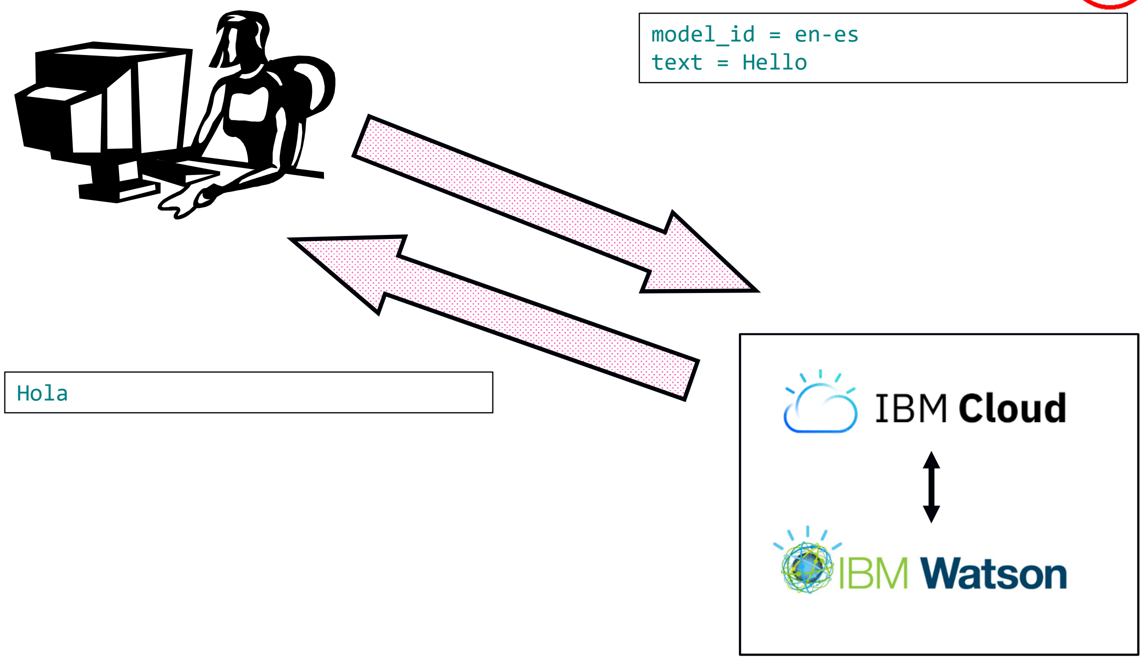
These are the fields you'll need to provide. (You can usually just take the defaults.)

- Service Name = Unique name for your newly created service. IBM Cloud will generate a name, but you can change it to whatever you like.
- Region to deploy in = Which IBM Cloud server location. Pick the one closest to you for best performance.
- Resource Group = Your organization.
- Tags = Space you set up when you signed up
- View Docs / View API Docs / Terms = Information for you

After creating

- it will take you to a getting started tutorial page to learn more
- You can access your already-set-up services in the Dashboard
  - Getting Started Tutorial
  - API Docs
  - Service credentials

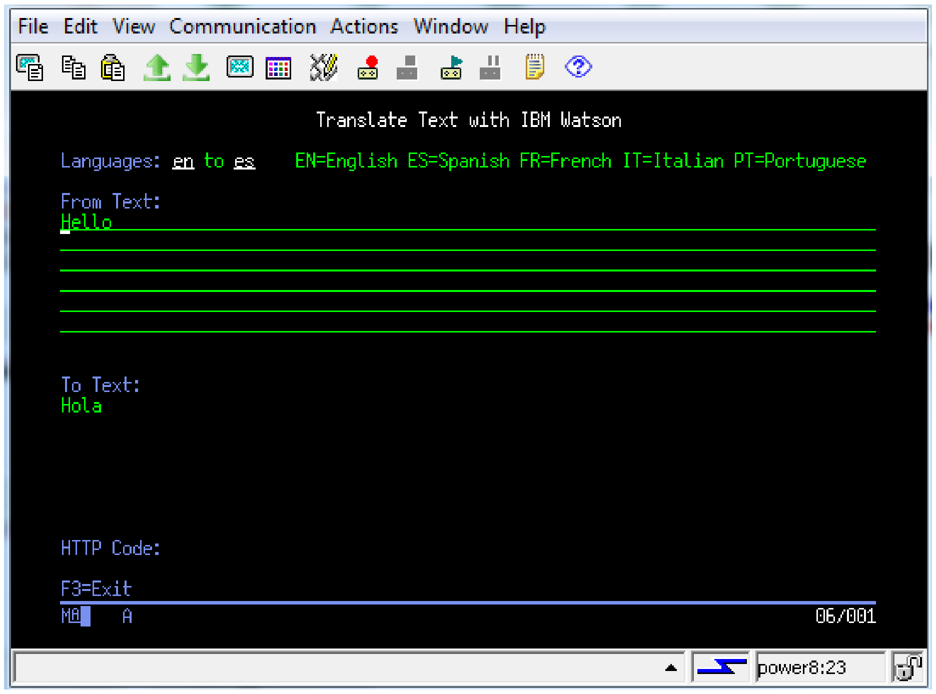
How Does an API Work?



```
model_id = en-es
text = Hello
```

```
Hola
```

Lets Do it In RPG!



```
                    Translate Text with IBM Watson

  Languages: en to es    EN=English ES=Spanish FR=French IT=Italian PT=Portuguese

  From Text:
  Hello






  To Text:
  Hola





  HTTP Code:

  F3=Exit
  MA     A                                                              06/001
```

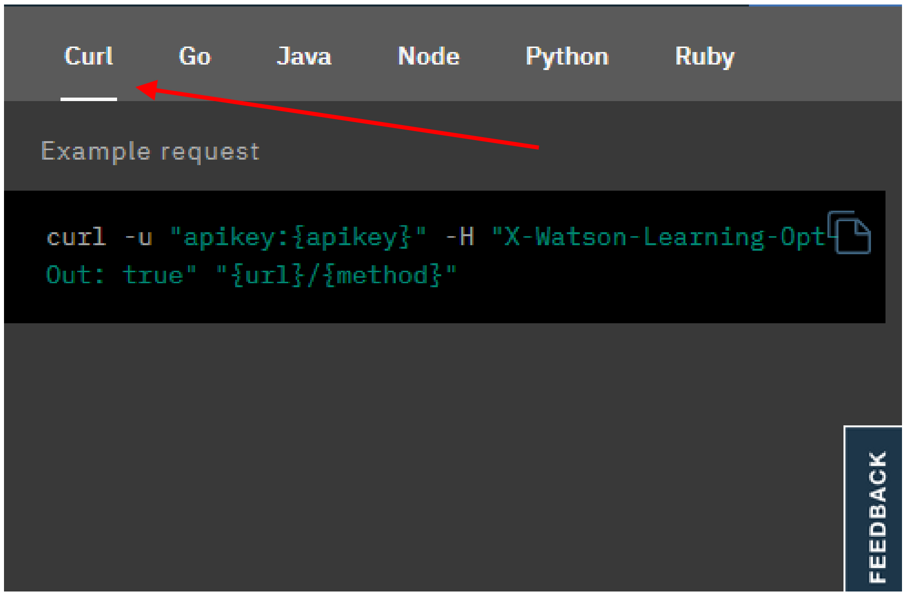
The Process in RPG

The process is basically the same!

- Show screen
- Get data from the screen and build strings
- Use HTTPAPI (or another HTTP tool) to send and receive
- Put the response string on the screen

Watson Docs – Use Curl

Docs are available for Curl, Node.js, Java and Python (but not RPG!)
Curl is the closest to plain HTTP, and easy to adapt to HTTPAPI



Example request

```
curl -u "apikey:{apikey}" -H "X-Watson-Learning-Opt-
Out: true" "{url}/{method}"
```

Watson Docs – Use Curl

Curl Is:

- command-line tool for HTTP and many other network protocols
- freely available (open source)
- most commonly found on Linux
- but, can run on IBM i in PASE (AIX version)
- or on Windows, Mac OS, Linux, FreeBSD, etc.

The Curl Home Page:
https://curl.haxx.se

The Curl Manual (to understand what the options do):
https://curl.haxx.se/docs/manpage.html

You will mainly use curl to test out and understand the Watson services, but will adapt them to HTTPAPI (or another HTTP tool for RPG) for production use. For that reason, I recommend running it on your PC.

About HTTPAPI

- HTTPAPI lets you make HTTP calls from RPG
- by Scott Klement, started in 2001 – but please use the latest!
- freely available (open source) at no charge
- very versatile
- very fast

Main routines that we want to use:

- http_req = routine for making an HTTP request (i.e. calling a URL)
- http_stmf = implified version of http_req when data sent/received is an IFS file
- http_string = simplified version of http_req when data send/received is in strings

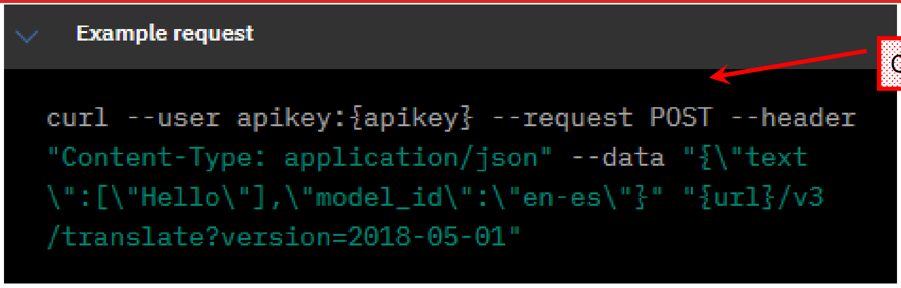
Other routines that complement the main ones:

- http_setOption = sets various options to control how HTTPAPI works
- http_setAuth = sets userid/password used by the HTTP protocol
- http_debug = generates a file in the IFS with diagnostic information
- http_error = retrieves the last error that occurred in an HTTPAPI routine

Adapting Curl to HTTPAPI

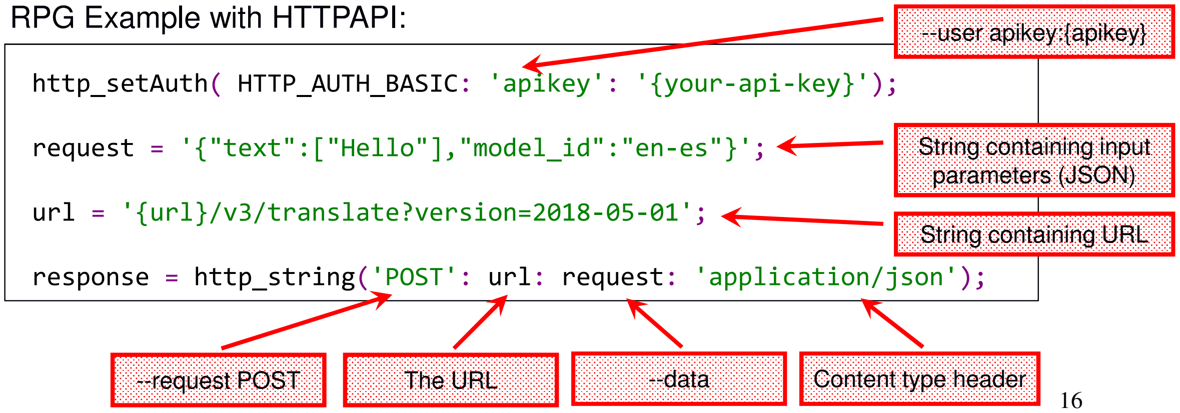
CURL OPTION	HTTPAPI Option
-X or --request with POST or GET	"type" parameter = POST or GET
No -X or –request specified	"type" parameter = GET
-u or --user with "user:pass"	Use HTTP_setAuth(HTTP_AUTH_BASIC: 'user': 'pass')
-H or --header with "content-type"	"contentType" parameter
-H or --header but not content-type	Use HTTP_xproc(HTTP_POINT_ADDL_HEADER) *See Translation with JSON example.*
-d or --data	SendStr or SendStmf parameter
-F or –form	SendStr or SendStmf encoded with "multipart/form-data" (MFD) *See Visual Recognition Example*
Use of @ to get from file	Use http_req or http_stmf with SendStmf parameter
URL parameter	URL parameter
Output	Use Response or Result Stmf/String parameters Or return value from http_string() API.

Adapting from Curl -- Example



```
curl --user apikey:{apikey} --request POST --header
"Content-Type: application/json" --data "{\"text
\":[\"Hello\"],\"model_id\":\"en-es\"}" "{url}/v3
/translate?version=2018-05-01"
```

RPG Example with HTTPAPI:



```
http_setAuth( HTTP_AUTH_BASIC: 'apikey': '{your-api-key}');

request = '{"text":["Hello"],"model_id":"en-es"}';

url = '{url}/v3/translate?version=2018-05-01';

response = http_string('POST': url: request: 'application/json');
```

Media Types (aka MIME types)

Various internet protocols use media types (also called "MIME" types) to identify the data type of something.

- text/plain = a plain text document
- application/json = a JSON document
- image/jpeg = a .JPG image or picture
- etc.

These HTTP headers can be used to communicate with an API about what sort of data we're sending and/or receiving back.

- content-type header = the type of the document we're sending
- accept header = the type of the document we want to receive back

Parameters Are Encoded in JSON

JSON is a data format used to make a string from program variables

- Here used to represent input/output parameters
- Made of strings, numbers, arrays and data structures
- Or a combination of the above!

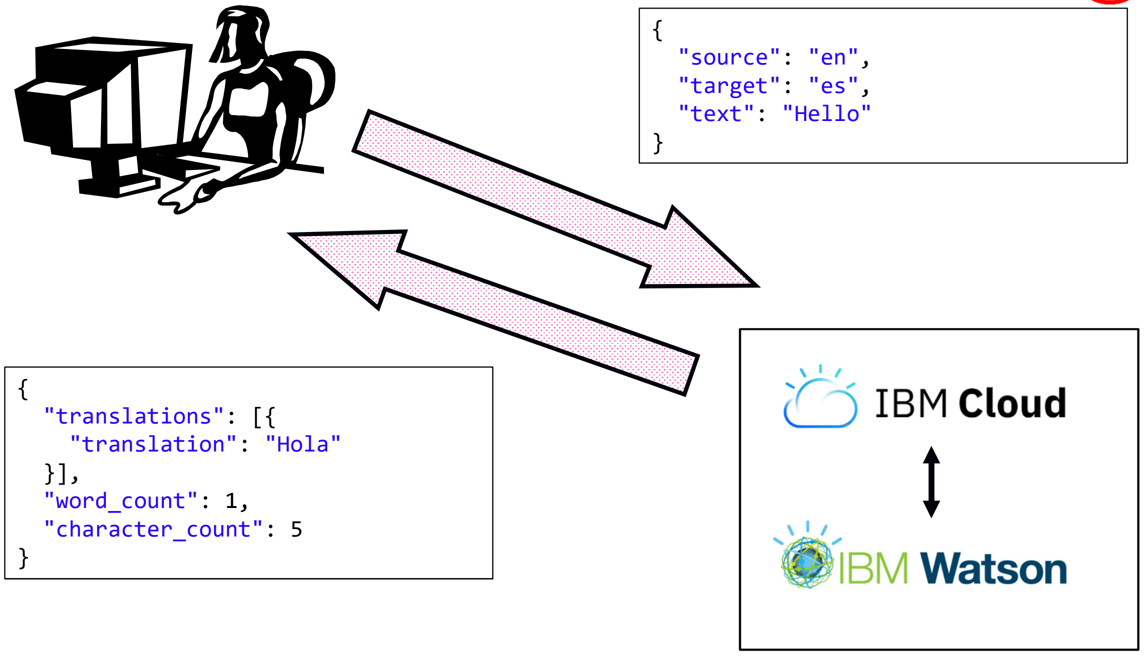
Example JSON:

```
{
  "source": "en",
  "target": "es",
  "text": ["Hello", "How Are You?"]
}
```

This is an alternative input parameter list, also found in Watson docs

- { } represents a data structure (called "object" in JSON terminology)
- This structure has two subfields named "model_id" and "text"
- Data In quotes is a string
- [ ] represents an array.   "text" is an array containing two elements, both strings.
- All one string. Computer doesn't care about indenting/linefeeds.

Think: Program Call



```
{
  "source": "en",
  "target": "es",
  "text": "Hello"
}
```

```
{
  "translations": [{
    "translation": "Hola"
  }],
  "word_count": 1,
  "character_count": 5
}
```

Generating JSON with YAJL

```
yajl_genOpen(*off);
yajl_beginObj();                        // {
yajl_addChar('source': fromLang);  //   "source": "en",
yajl_addChar('target': toLang);    //   "target": "es",
yajl_addChar('text': fromText );   //   "text": "Hello"
yajl_endObj();                          // }

request = yajl_copyBufStr();
yajl_genClose();
```

Very quick introduction to the YAJL tools for generating JSON documents:

- yajl_genOpen / yajl_genClose start/stop the JSON generator
- yajl_beginObj / yajl_endObj create a JSON object (the stuff in curly braces)
- yajl_addChar adds a character string to the open object
- yajl_addNum, yajl_addBool, yajl_beginArray, yajl_endArray also available
- yajl_copyBufStr copies the generated JSON buffer to an RPG string

Reading JSON with DATA-INTO

```
dcl-ds result qualified;
  dcl-ds translations dim(1);
    translation varchar(1000);
  end-ds;
  word_count int(10);
  character_count int(10);
end-ds;
```

```
{
  "translations": [{
    "translation": "{string}"
  }],
  "word_count": {number},
  "character_count": {number}
}
```

You must declare an RPG variable that matches the JSON

- { } JSON objects match RPG DS (dcl-ds)
- [ ] JSON arrays match RPG arrays (DIM)
- Field names must be the same
- JSON Strings (quoted) match RPG char, varchar, ucs2, etc
- JSON Numbers (unquoted) match RPG packed, zoned, integer, etc.

Then use DATA-INTO opcode:

```
DATA-INTO result %DATA(response) %PARSER('YAJLINTO');
```

YAJLINTO is a parser for data-into that comes with the YAJL download.

Translate with JSON (1 of 5)

To put all of these concepts together, here's the full RPG code for the translate example, using JSON rather than plain text.

```
**free
ctl-opt option(*srcstmt) dftactGrp(*no)
        bnddir('HTTPAPI':'YAJL');

/copy httpapi_h
/copy yajl_h

dcl-f WATSONTR3D workstn indds(dspf);

dcl-Ds dspf qualified;
   F3Exit ind pos(3);
end-Ds;

dcl-c UPPER 'ENESFRITPT';
dcl-c lower 'enesfritpt';

fromLang = 'en';
toLang   = 'es';
```

BNDDIR is used to bind your program to the tools

Copybooks contain the definitions we'll need to call the HTTPAPI and YAJL routines

Translate with JSON (2 of 5)

Main loop controls the flow of the program, repeating the screen until F3 key is pressed.

```
dou dspf.F3Exit = *on;

   exfmt screen1;
   if dspf.F3exit = *on;
      leave;
   endif;

   fromLang = %xlate(UPPER:lower:fromLang);
   toLang   = %xlate(UPPER:lower:toLang);
   toText = translate( fromLang: toLang: %trim(fromText) );

enddo;

*inlr = *on;
return;
```

the translate procedure is what actually calls Watson

Translate with JSON (3 of 5)

```
dcl-proc translate;

   dcl-pi *n varchar(1000);
      fromLang char(2)        const;
      tolang   char(2)        const;
      fromText varchar(1000)  const;
   end-pi;

   dcl-s url      varchar(2000);
   dcl-s request  varchar(2000);
   dcl-s response varchar(5000);

   dcl-ds result qualified;
     dcl-ds translations dim(1);
        translation varchar(1000);
     end-ds;
     word_count int(10);
     character_count int(10);
   end-ds;
```

Most of this slide is just ordinary RPG definitions

Data structure must match the JSON format for the output parameters.

Translate with JSON (4 of 5)

```
yajl_genOpen(*off);
yajl_beginObj();                      // {
  yajl_addChar('source': fromLang);  //   "source": "en",
  yajl_addChar('target': toLang);    //   "target": "fr",
  yajl_beginArray('text');           //   "text": [
    yajl_addChar(fromText );         //      "String here"
  yajl_endArray();                   //   ]
yajl_endObj();                        // }

request = yajl_copyBufStr();

yajl_genClose();
```

Generate the JSON document to send

Put the JSON data into a varchar variable named "request"

Clean up the JSON data stored inside of YAJL

Translate with JSON (5 of 5)

```
url = 'https://gateway.watsonplatform.net/language-translator/api'
    + '/v3/translate?version=2018-05-01';

monitor;
   response = http_string('POST': url: request: 'application/json');
on-error;
   httpcode = http_error();
endmon;

DATA-INTO result %DATA(response) %PARSER('YAJLINTO');

return result.translations(1).translation;

end-Proc;
```

Send 'request' (input) and get back 'response' (output)

Load output into 'result' using data-into

Return the first string translation back to mainline of program

Got the Idea?

You now know the basics of how to call the Watson services from RPG.

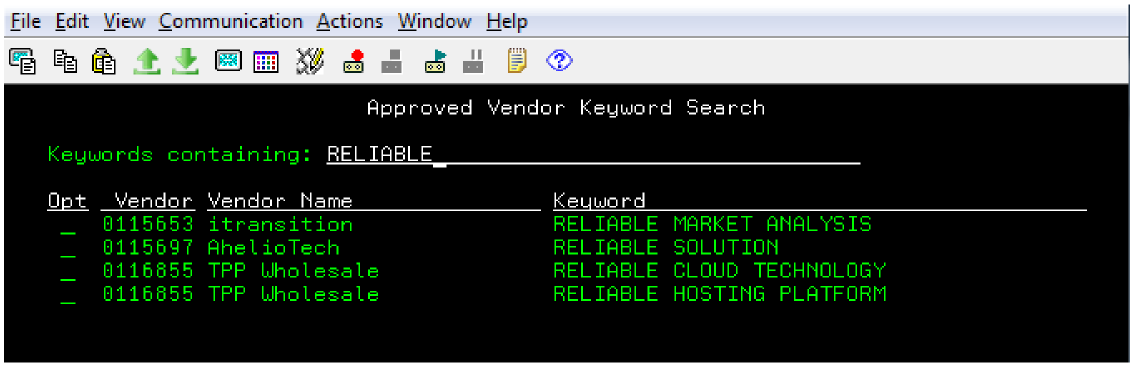
- Find the service you want
- Create it
- Read the documentation for that service
- Use "curl" examples
- Adapt the "curl" examples to HTTPAPI

Following the same techniques will allow you to call any of Watson's services!

Example of Finding Approved Vendors

This Example uses Watson's *Natural Language Understanding* API

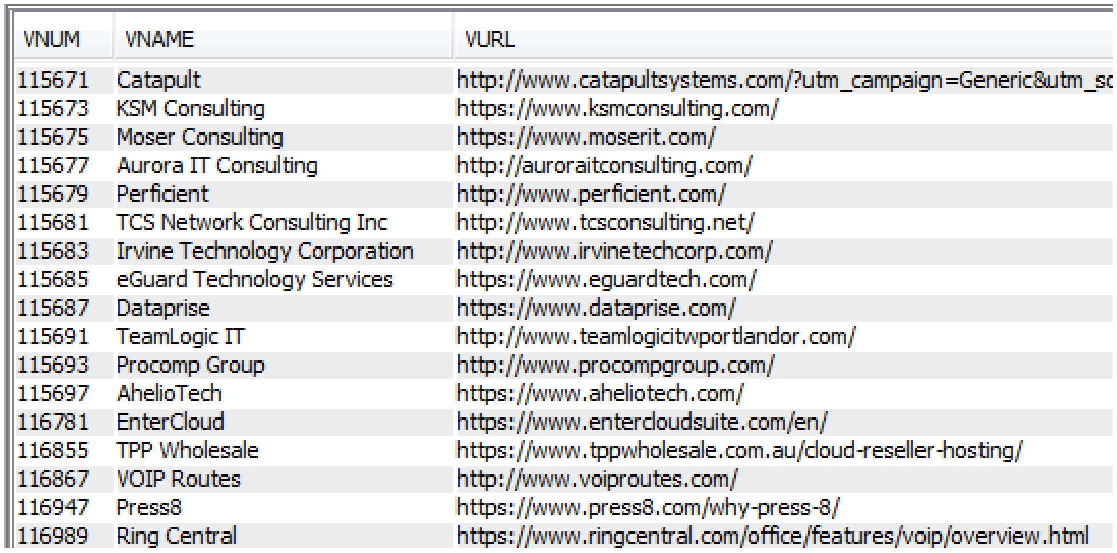
- Loop through a table (PF) with approved vendor's web sites
- Ask Watson to analyze them and find keywords on their site
- Put the keywords in a file
- Allow the user to query the file by keyword



```
                     Approved Vendor Keyword Search

  Keywords containing: RELIABLE_

  Opt  Vendor  Vendor Name                  Keyword
   _   0115653 itransition                  RELIABLE MARKET ANALYSIS
   _   0115697 AhelioTech                   RELIABLE SOLUTION
   _   0116855 TPP Wholesale                RELIABLE CLOUD TECHNOLOGY
   _   0116855 TPP Wholesale                RELIABLE HOSTING PLATFORM
```

The AVLIST Table

This is what the "AVLIST" (approved vendor list) table (PF) looks like (excerpt)



VNUM	VNAME	VURL
115671	Catapult	http://www.catapultsystems.com/?utm_campaign=Generic&utm_sc
115673	KSM Consulting	https://www.ksmconsulting.com/
115675	Moser Consulting	https://www.moserit.com/
115677	Aurora IT Consulting	http://auroraitconsulting.com/
115679	Perficient	http://www.perficient.com/
115681	TCS Network Consulting Inc	http://www.tcsconsulting.net/
115683	Irvine Technology Corporation	http://www.irvinetechcorp.com/
115685	eGuard Technology Services	https://www.eguardtech.com/
115687	Dataprise	https://www.dataprise.com/
115691	TeamLogic IT	http://www.teamlogicitwportlandor.com/
115693	Procomp Group	http://www.procompgroup.com/
115697	AhelioTech	https://www.aheliotech.com/
116781	EnterCloud	https://www.entercloudsuite.com/en/
116855	TPP Wholesale	https://www.tppwholesale.com.au/cloud-reseller-hosting/
116867	VOIP Routes	http://www.voiproutes.com/
116947	Press8	https://www.press8.com/why-press-8/
116989	Ring Central	https://www.ringcentral.com/office/features/voip/overview.html

Natural Language Understanding (1 of 5)

This is the main loop that controls the program to analyze the vendor's sites:

```
exec SQL declare C1 cursor for
   select * from AVLIST;

exec SQL open C1;

exec SQL fetch next from C1 into :AVLIST;
dow sqlstt='00000';

   request = createJsonRequest(AVLIST);

   monitor;
      response = http_string('POST': url: request: 'application/json');
   on-error;
      response = '';
   endmon;

   extractKeywords(AVLIST.VNUM: response);

   exec SQL fetch next from C1 into :AVLIST;
enddo;

exec SQL close C1;
```

Natural Language Understanding (2 of 5)

Create the JSON document that is sent to Watson to tell it what to do.

```
dcl-proc createJsonRequest;

   dcl-pi *N varchar(1000);
      VEND likeds(AVLIST) const;
   end-pi;

   dcl-s json varchar(1000);

   yajl_genOpen(*off);
   yajl_beginObj();                          // {
     yajl_addChar('url': %trim(VEND.VURL));  //    "url": "http://example.com",
     yajl_beginObj('features');              //    "features" : {
       yajl_beginObj('keywords');            //       "keywords": {
       yajl_endObj();                        //       }
     yajl_endObj();                          //    }
   yajl_endObj();                            //

   json = yajl_copyBufStr();
   yajl_genClose();

   return json;
end-proc;
```

Natural Language Understanding (3 of 5)

The "extractKeywords" subprocedure interprets the JSON document that Watson sent back, gets the keywords that it found, and writes those keywords to a second table named AVKEYWORDS.

This AVKEYWORDS table (PF) can then be searched by the user.

```
dcl-proc extractKeywords;

   dcl-pi *N;
      vnum packed(7: 0) value;
      json varchar(100000) const;
   end-pi;

   dcl-s text char(40);
   dcl-s i int(10);
   dcl-s success ind inz(*on);

   dcl-ds result qualified;
      num_keywords int(10);
      dcl-ds keywords dim(1000);
         text varchar(40);
         relevance packed(8: 6);
      end-ds;
   end-ds;
```

num_keywords is using an RPG feature to count the number of array elements, so does not appear in the JSON

```
{
  "keywords": [
    {
      "text": "{string}",
      "relevance": {number}
    }
  ]
}
```

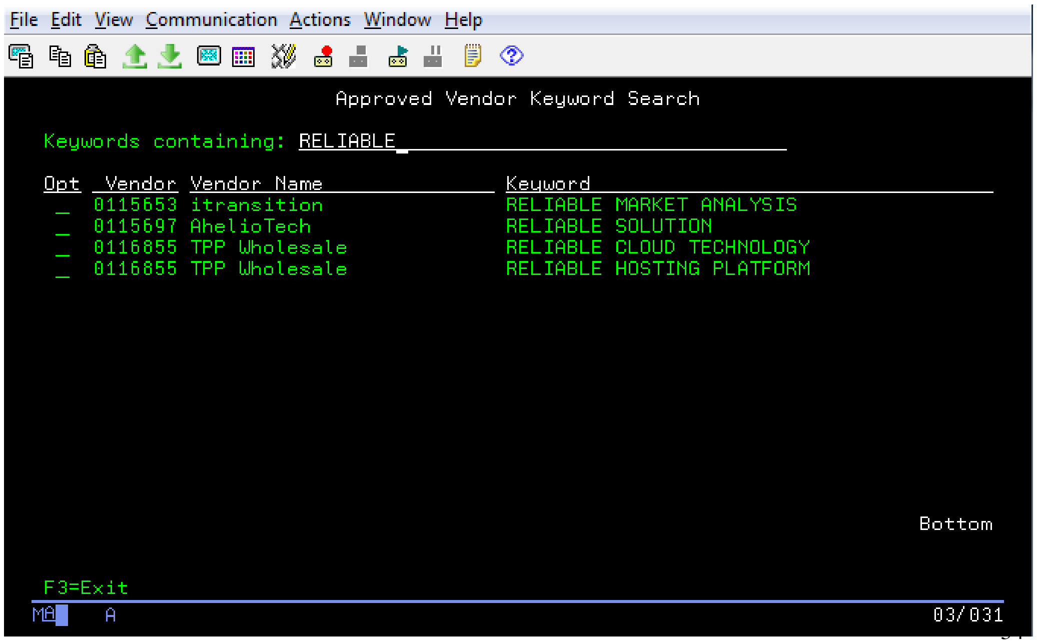
Natural Language Understanding (4 of 5)

```
monitor;
     data-into result %DATA(response: 'countprefix=num_ allowextra=yes')
                      %PARSER('YAJLINTO');
  on-error;
     success = *off;
  endmon;

  if success;
     for i = 1 to result.num_keywords;
        text = result.keywords(i).text;
        if result.keywords(i).relevance > 0.35;
           exec SQL insert into AVKEYWORDS values(:vnum, UPPER(:text));
        endif;
     endfor;
  endif;

end-proc;
```

Only use keywords if Watson is at least 35% confident

Natural Language Understanding (5 of 5)

A second program uses traditional RPG techniques (code not shown – it's just normal RPG) to search the AVKEYWORDS table that we just populated.



```
                          Approved Vendor Keyword Search

  Keywords containing: RELIABLE_

  Opt  Vendor Vendor Name                 Keyword
   _   0115653 itransition                RELIABLE MARKET ANALYSIS
   _   0115697 AhelioTech                 RELIABLE SOLUTION
   _   0116855 TPP Wholesale              RELIABLE CLOUD TECHNOLOGY
   _   0116855 TPP Wholesale              RELIABLE HOSTING PLATFORM

                                                                  Bottom

  F3=Exit
 MA      A                                                        03/031
```

Visual Recognition API

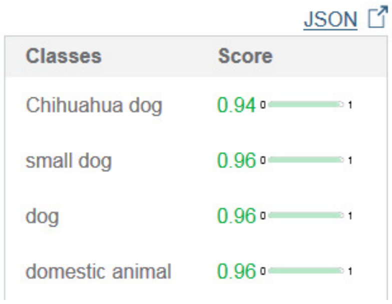
Watson's Visual Recognition can "understand" the contents of photographs or video frames. It seeks to answer two questions:

- What is in this image?
- Are there other, similar, images?

There is setup involved. You must "train" Watson in how to recognize a given subject.

Once trained, however, it can recognize the same subject in any picture, even from different angles.

Training the Visual Recognition Tool


JSON

Classes	Score
Chihuahua dog	0.94
small dog	0.96
dog	0.96
domestic animal	0.96

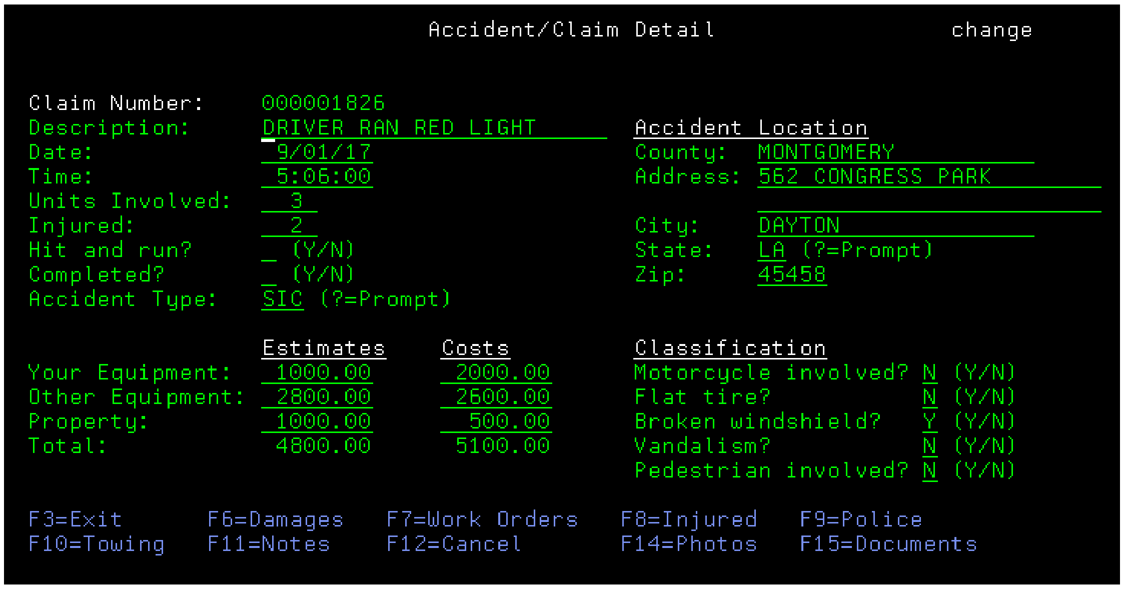
Training involves:

- Upload .ZIP files with matching images.
- Upload .ZIP file(s) with "negative" matches (to teach Watson what not to consider a match)

There is a tool on IBM Cloud that you can use to train the API. (No need to write code to do that if you don't want to!)

Insurance Claim Example (Background Info)

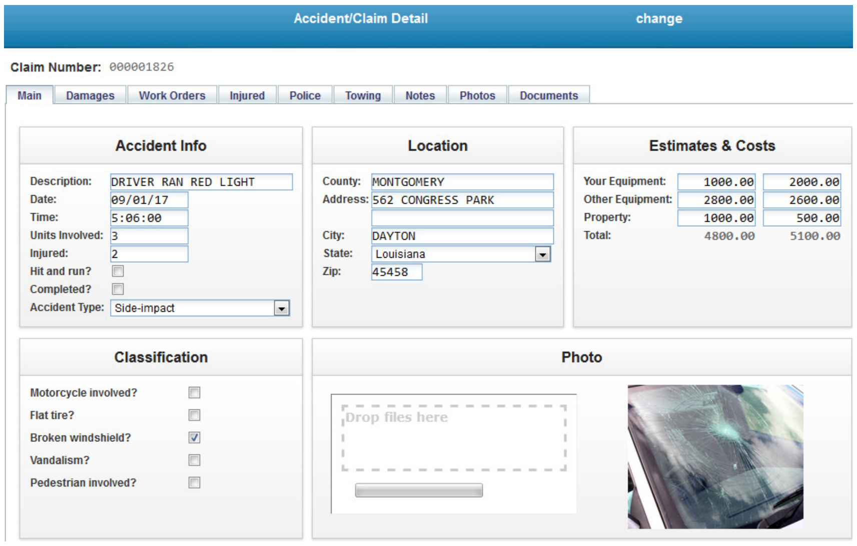
This example started with an old, green-screen, RPG program for entering insurance claims (such as car accidents)



```
                              Accident/Claim Detail                     change

 Claim Number:     000001826
 Description:      DRIVER RAN RED LIGHT         Accident Location
 Date:              9/01/17                     County:  MONTGOMERY
 Time:              5:06:00                     Address: 562 CONGRESS PARK
 Units Involved:     3
 Injured:            2                          City:    DAYTON
 Hit and run?      _  (Y/N)                     State:   LA (?=Prompt)
 Completed?        _  (Y/N)                     Zip:     45458
 Accident Type:    SIC (?=Prompt)

                   Estimates     Costs          Classification
 Your Equipment:    1000.00      2000.00        Motorcycle involved? N (Y/N)
 Other Equipment:   2800.00      2600.00        Flat tire?           N (Y/N)
 Property:          1000.00       500.00        Broken windshield?   Y (Y/N)
 Total:             4800.00      5100.00        Vandalism?           N (Y/N)
                                                Pedestrian involved? N (Y/N)

 F3=Exit       F6=Damages    F7=Work Orders   F8=Injured    F9=Police
 F10=Towing    F11=Notes     F12=Cancel       F14=Photos    F15=Documents
```

Insurance Claim Example (Background Info)

Years later (but still awhile back) this application was converted to a web-based GUI using Profound Logic's tools. RPG Open Access was used so the RPG code didn't have to change, though a small amount of code was added to allow picture uploads.



Accident/Claim Detail change

Claim Number: 000001826

Accident Info

Description: DRIVER RAN RED LIGHT
Date: 09/01/17
Time: 5:06:00
Units Involved: 3
Injured: 2
Hit and run? ☐
Completed? ☐
Accident Type: Side-impact

Location

County: MONTGOMERY
Address: 562 CONGRESS PARK
City: DAYTON
State: Louisiana
Zip: 45458

Estimates & Costs

Your Equipment:	1000.00	2000.00
Other Equipment:	2800.00	2600.00
Property:	1000.00	500.00
Total:	4800.00	5100.00

Classification

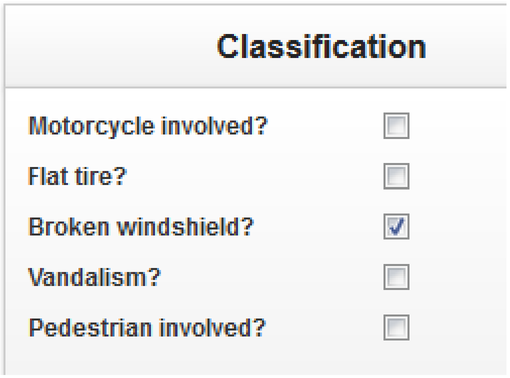
Motorcycle involved? ☐
Flat tire? ☐
Broken windshield? ☑
Vandalism? ☐
Pedestrian involved? ☐

Photo

Insurance Claim Example (Background Info)

We decided to use Watson to help set the "Classication" of each claim.

- Trained Watson with images to recognize each classification category
- Now, when a user adds a new picture (or changes existing one)
- Watson figures out the right "classification" of the claim.



Classification

Motorcycle involved?	☐
Flat tire?	☐
Broken windshield?	✓
Vandalism?	☐
Pedestrian involved?	☐

What Watson Determines

We wrote a routine named watson_classify for our RPG program. It accepts the pathname to the photograph (in the IFS) and returns the following data structure:

```
D classify_t      ds                  qualified
D                                     template
D   code                      10i 0 inz(1)
D   errMsg                   500a   varying inz('')
D   class                    256a   varying inz('')
D   score                      9p 7 inz(0)

D obj             ds                  likeds(classify_t)
```

- "code" and "errMsg" are used to report an error (if any)
- "class" is the classification determined by Watson
- "score" is how confident (from 0=unlikely to 1=completely positive) Watson was

Incorporating Into Existing Code

The existing fixed format code was modified to call the Watson Visual Recognition "classify" API when the image was changed, and based on Watson's "first" choice (most likely classification) it sets the classification field to Y or N

```
     C                     If        UploadInfo = '001'
     C                     Eval      done = *Off

      // -------------------------------------------------------
      // New Code for recognizing image:
     C                     eval      obj = watson_classify(imagefile)
     C                     if        obj.score > 0.75
     C
     C                     select
     C                     when      obj.class = 'motorcycleaccident'
     C                     eval      cmmotor = 'Y'
     C                     when      obj.class = 'brokenwinshield'
     C                     eval      cmbrokenw = 'Y'
        ...etc...
```

If Watson wasn't 75% sure, we ignored it's classification (leaving it to the user)

Visual Recognition RPG (1 of 6)

```
dcl-proc watson_classify;

   dcl-pi *n likeds(classify_t);
      imageName varchar(256) const;
   end-pi;

   dcl-c WATSON_API_KEY 'the api key from IBM Cloud is put here';

   dcl-s imagePath   varchar(256);
   dcl-s params      varchar(256);
   dcl-s form        pointer;
   dcl-s contentType char(64);
   dcl-s tempFile    varchar(256);
   dcl-s rc          int(10);
   dcl-s docNode     like(yajl_val);
   dcl-s topClass    like(yajl_val);
   dcl-s node        like(yajl_val);
   dcl-s response    varchar(100000);
   dcl-s errMsg      varchar(500);
   dcl-s url         varchar(500);

   dcl-ds result likeds(classify_t) inz;
```

Visual Recognition RPG (2 of 6)

```
http_debug(*on: '/tmp/watson-claim15r.txt');

imagePath = '/www/profoundui/htdocs/profoundui/userdata/'
          + 'images/claims/' + %trim(imageName);

// Create a JSON document containing the
//  parameters to the "classify" API:

yajl_genOpen(*off);
yajl_beginObj();                             // {
yajl_beginArray('classifier_ids');           //    "classifier_ids": [
yajl_addChar('insuranceclaims_1650517727');  //        "id-goes-here"
yajl_endArray();                             //    ]
yajl_endObj();                               // }
params = yajl_copyBufStr();
yajl_genClose();
```

The input JSON document is very simple – it just tells Watson which "classifier" to use (i.e. which set of data that we trained Watson with)

Visual Recognition RPG (3 of 6)

```
// Create a multipart/form-data form (like curl -F switch)
// to contain both the image and the parameters
// (this is created in a temporary IFS file)

tempFile = http_tempfile();
form = http_mfd_encoder_open( tempFile: contentType );
http_mfd_encoder_addstmf( form
                        : 'images_file'
                        : imagePath
                        : 'image/jpeg' );
http_mfd_encoder_addvar_s( form: 'parameters': params );
http_mfd_encoder_close(form);
```

The image is uploaded using a "multi part form", like a web browser would use.

- One part contains the input JSON document (from last slide)
- The other part contains the image.

HTTPAPI's multi-part form data (MFD) tool creates this form in a temporary IFS file. Although the images in the insurance claims are small, HTTPAPI has the capacity to handle multiple gigabytes of data, so keeping the form in memory isn't practical.

Visual Recognition RPG (4 of 6)

```
 url = 'https://gateway.watsonplatform.net'
     + '/visual-recognition/api/v3/classify'
     + '?version=2018-03-19';

 http_setAuth( HTTP_AUTH_BASIC: 'apikey': WATSON_API_KEY );

 rc = http_req( 'POST'
              : url
              : *omit: response
              : tempFile: *omit
              : %trim(contentType) );

// delete temporary file -- no longer needed.
unlink(tempFile);
```

Since the input is an IFS file, but I wanted the output to be returned as a string, I used the http_req() routine.

There are two parameters for "response data", for string and file, respectively. Also two for "send data", a string and a file. One send option and one response option must be set to *omit.

Response JSON Document

```
{
  "custom_classes":5,
  "images":[
    {
      "classifiers":[
        {
          "classes":[
            {
              "class":"brokenwinshield",
              "score":0.976408
            }
          ],
          "classifier_id":"insuranceclaims_1650517727",
          "name":"insurance-claims"
        }
      ],
      "image":"image path name here"
    }
  ],
  "images_processed":1
}
```

We only upload one image at a time, so there will never be more than one entry in the images array.

Only asked for one classifier (trained databased) so "classifiers" will only have one array entry.

Watson might see more than one class. But the "top pick" will be first.

Conclusion: *We want the first array entry inside the first array entry of the first array entry!*

Visual Recognition RPG (5 of 6)

```
dcl-ds watsonData qualified;
  dcl-ds images dim(1);
    num_classifiers int(10);
    dcl-ds classifiers dim(1);
      num_classes   int(10);
      dcl-ds classes dim(10);
        class varchar(100);
        score packed(7: 3);
      end-ds;
      classifier_id varchar(100);
      name          varchar(100);
  end-ds;
end-ds;
```

```
{
  "images":[{

    "classifiers":[{

      "classes":[{
         "class":"{string}",
         "score": {number}
      }],

      "classifier_id":"{string}",
      "name":"{string}"
    }]
  }]
}
```

Visual Recognition RPG (6 of 6)

```
  monitor;
     data-into watsonData %DATA( response
                               : 'case=convert +
                                  countprefix=num_ +
                                  allowextra=yes')
                          %PARSER('YAJLINTO');

     if watsonData.images(1).num_classifiers >= 1
        and watsonData.images(1).classifiers(1).num_classes >= 1;
           result.class = watsonData.images(1)
                          .classifiers(1).classes(1).class;
           result.score = watsonData.images(1)
                          .classifiers(1).classes(1).score;
     endif;
  on-error;
     result.code = -1;
     result.errMsg = 'JSON Parse failed';
  endmon;

  return result;

end-proc;
```

Conclusion

This presentation barely scratches the surface of what Watson can do!

- Cognitive Computing
- A computer that can "think"!
- Understand human language better than ever before
- Understand human photographs better than ever before
- Can search and understand massive volumes of (unstructured) documents and pictures and discover trends, patterns, etc.
- Best of all, anyone can use it – from RPG on IBM i!
- It's just a web service (API) call!

This Presentation

You can download a PDF copy of this presentation and the sample code that I used from

http://www.scottklement.com/presentations/

Thank you!